Parsing Linear Context-Free Rewriting Systems with Fast Matrix Multiplication

Citation for published version:
Cohen, S & Gildea, D 2016, 'Parsing Linear Context-Free Rewriting Systems with Fast Matrix Multiplication', *Computational Linguistics*, vol. 42, no. 3, pp. 421-455. https://doi.org/10.1162/COLI_a_00254

Digital Object Identifier (DOI):
10.1162/COLI_a_00254

Published In:
Computational Linguistics

Parsing Linear Context-Free Rewriting Systems with Fast Matrix Multiplication

Shay B. Cohen *
University of Edinburgh

Daniel Gildea **
University of Rochester

We describe a matrix multiplication recognition algorithm for a subset of binary linear context-free rewriting systems (LCFRS) with running time $O(n^{\omega d})$ where $M(m) = O(m^{\omega})$ is the running time for $m \times m$ matrix multiplication and d is the "contact rank" of the LCFRS – the maximal number of combination and non-combination points that appear in the grammar rules. We also show that this algorithm can be used as a subroutine to get a recognition algorithm for general binary LCFRS with running time $O(n^{\omega d+1})$. The currently best known ω is smaller than 2.38. Our result provides another proof for the best known result for parsing mildly context sensitive formalisms such as combinatory categorial grammars, head grammars, linear indexed grammars, and tree adjoining grammars, which can be parsed in time $O(n^{4.76})$. It also shows that inversion transduction grammars can be parsed in time $O(n^{5.76})$. In addition, binary LCFRS subsumes many other formalisms and types of grammars, for some of which we also improve the asymptotic complexity of parsing.

1. Introduction

The problem of **grammar recognition** is a decision problem that requires to determine whether a string belongs to a language induced by a grammar. For context-free grammars, recognition can be done using parsing algorithms such as the CKY algorithm (Kasami 1965; Younger 1967; Cocke and Schwartz 1970) or the Earley algorithm (Earley 1970). The asymptotic complexity of these chart parsing algorithms is cubic in the length of the sentence.

In a major breakthrough, Valiant (1975) showed that context-free grammar recognition is no more complex than Boolean matrix multiplication for a matrix of size $m \times m$ where m is linear in the length of the sentence, n. With current state-of-the-art results in matrix multiplication, this means that CFG recognition can be done with an asymptotic complexity of $O(n^{2.38})$.

In this paper, we show that the problem of linear context-free rewriting system recognition can also be reduced to Boolean matrix multiplication. Current chart parsing algorithms for binary LCFRS have an asymptotic complexity of $O(n^{3f})$, where f is the maximal fan-out of the grammar.[1] Our algorithm takes time to $O(n^{\omega d})$, for a constant d which is a function of the grammar (and not the input string), and where the complexity of $n \times n$ matrix multiplication is $M(n) = O(n^{\omega})$. The parameter d can be as small as f, meaning that we reduce parsing complexity from $O(n^{3f})$ to $O(n^{\omega f})$, and that, in general, the savings in the exponent is larger for more complex grammars.

* School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, United Kingdom. E-mail: `scohen@inf.ed.ac.uk`.

** Department of Computer Science, University of Rochester, Rochester, NY 14627, United States. E-mail: `gildea@cs.rochester.edu`.

1 Without placing a bound on f, the problem of recognition linear context-free rewriting systems is NP-hard (Satta 1992).

LCFRS is a broad family of grammars. As such, we are able to support the findings of Rajasekaran and Yooseph (1998), who showed that tree adjoining grammar recognition can be done in time $O(M(n^2)) = O(n^{4.76})$ (TAG can be reduced to LCFRS with $d = 2$). As a result, combinatory categorial grammars, head grammars and linear indexed grammars can be recognized in time $O(M(n^2))$. In addition, we show that inversion transduction grammars (Wu 1997, ITGs) can be parsed in time $O(nM(n^2)) = O(n^{5.76})$, improving the best asymptotic complexity previously known for ITGs.

Matrix Multiplication State of the Art. Our algorithm reduces the problem of LCFRS parsing to Boolean matrix multiplication. Let $M(n)$ be the complexity of multiplying two such $n \times n$ matrices. These matrices can be naïvely multiplied in $O(n^3)$ time by computing for each output cell the dot product between the corresponding row and column in the input matrices (each such product is an $O(n)$ operation). Strassen (1969) discovered a way to do the same multiplication in $O(n^{2.8704})$ time – his algorithm is a divide and conquer algorithm that eventually uses only 7 operations (instead of 8) to multiply 2×2 matrices.

With this discovery, there have been many attempts to further reduce the complexity of matrix multiplication, relying on principles similar to Strassen's method: a reduction in the number of operations it takes to multiply sub-matrices of the original matrices to be multiplied. While Strassen's algorithm is a practical algorithm, not all of its improvements lead to practical algorithms. Coppersmith and Winograd (1987) discovered an algorithm that has the asymptotic complexity of $O(n^{2.375477})$, with a large impractical constant lurking in the O-notation. Others have slightly improved their algorithm, and currently there is an algorithm for matrix multiplication with $M(n) = O(n^\omega)$ such that $\omega = 2.3728639$ (Le Gall 2014). It is known that $M(n) = \Omega(n^2 \log n)$ (Raz 2002).

Main Result. Our main result is a matrix multiplication algorithm for *unbalanced* binary LCFRS with asymptotic complexity $M(n^d) = O(n^{\omega d})$ where d is the maximal number of combination points in all grammar rules. The constant d can be easily determined from the grammar at hand:

$$d = \max_{a \to b\, c} \max \left\{ \begin{array}{c} \varphi(a) + \varphi(b) - \varphi(c), \\ \varphi(a) - \varphi(b) + \varphi(c), \\ -\varphi(a) + \varphi(b) + \varphi(c) \end{array} \right\}. \quad (1)$$

where $a \to b\, c$ ranges over rules in the grammar and $\varphi(a)$ is the fan-out of nonterminal a. The notion of unbalanced grammars is introduced in §4.4, and it is a condition on the set of LCFRS grammar rules that is satisfied with many practical grammars. In cases where the grammar is balanced, our algorithm can be used as a sub-routine so that it parses the binary LCFRS in time $O(n^{\omega d + 1})$. A similar procedure was applied by Nakanishi et al. (1998) for multiple component context-free grammars. See more discussion of this in §7.4.

Our results focus on the asymptotic complexity as a function of *string length*. We do not give explicit grammar constants. For other work that focuses on reducing the grammar constant in parsing, see for example Eisner and Satta (1999), Dunlop, Bodenstab, and Roark (2010), Cohen, Satta, and Collins (2013). For a discussion of the optimality of the grammar constants in Valiant's algorithm, see for example Abboud, Backurs, and Williams (2015).

2. Background and Notation

For an integer n, let $[n]$ denote the set of integers $\{1, \ldots, n\}$. Let $[n]_0 = [n] \cup \{0\}$. For a set A, we denote by A^+ the set of all sequences of length 1 or more of elements from A.

A span is a pair of integers denoting left and right endpoints for a substring in a larger string. The endpoints are placed in the "spaces" between the symbols in a string. For example, the span $(0, 3)$ spans the first three symbols in the string. For a string of length n, the set of potential endpoints is $[n]_0$.

We say a sequence of pairs of integers (n_1, n_2),$(n_3, n_4), \ldots, (n_{K-1}, n_K)$ is "linked" if $n_r = n_{r+1}$ for any even integer r in the set $[K-1]$. One can imagine a linked sequence of pairs of integers denoting spans in a string (each pair being a span) that are concatenated together to create one larger contiguous span (n_1, n_K). To concatenate them, we need the right endpoint of a given pair to be identical to the left endpoint of the next pair.

We turn now to give a succinct definition for binary LCFRS. For more details about LCFRS and their relationship to other grammar formalisms, see Kallmeyer (2010). A binary LCFRS is a tuple $(\mathcal{N}, \mathcal{T}, \mathcal{R}, \mathcal{V}, \varphi, S)$ such that:

- $\mathcal{N}$ is the set of non-terminal symbols in the grammar.
- $\mathcal{T}$ is the set of terminal symbols in the grammar. We assume $\mathcal{N} \cap \mathcal{T} = \emptyset$.
- φ is a function specifying a fixed fan-out for each non-terminal ($\varphi\colon \mathcal{N} \to \mathbb{N}$).
- $\mathcal{V}$ is a set of variables (we denote a variable by the letter y, potentially with subscripts or superscripts). We assume $\mathcal{N} \cap \mathcal{V} = \mathcal{T} \cap \mathcal{V} = \emptyset$.
- $\mathcal{R}$ is a set of production rules such that each rule in $\mathcal{R}$ has one of the following forms: (a) $a[\alpha] \to b[\beta]\, c[\gamma]$ such that $a, b, c \in \mathcal{N}$, $\alpha \in (\mathcal{V}^+)^{\varphi(a)}$, $\beta \in \mathcal{V}^{\varphi(b)}$, and $\gamma \in \mathcal{V}^{\varphi(c)}$. Each variable in $\mathcal{V}$ appears at most once in the union of variables in β and γ, and all variables in β and γ appear exactly once in α. Without loss of generality, we assume that $\alpha_1 = \beta_1$. (b) $a[\alpha] \to x_1[\beta_1], \ldots, x_{\varphi(a)}[\beta_{\varphi(a)}]$ where $\alpha \in \mathcal{V}^{\varphi(a)}$, $x_i \in \mathcal{T}$ and $\beta_i \in \mathcal{V}$, and all variables in β_i appear exactly once in α (in some order).
- $S \in \mathcal{N}$ is a start symbol. Without loss of generality, we assume $\varphi(S) = 1$.

The **fan-out** of a nonterminal is the number of spans in the input sentence that it covers, and the variables in a production specify how the spans of the r.h.s. nonterminals combine to produce the spans of l.h.s. nonterminal. For example, with context-free grammars, rules have the form:

$$a[xy] \to b[x]\, c[y]$$

indicating that b and c each have one span, and are concatenated in order to form a. As another example, consider that a binary tree adjoining grammar can be represented as a binary LCFRS (Vijay-Shanker and Weir 1994) with fan-out 2. Figure 1 demonstrates how the adjunction operation is done with binary LCFRS. Each gray block denotes a span, and the adjunction operator takes the first span of nonterminal b and concatenates it to the first span of nonterminal c (to get the first span of a), and then takes the second span of c and concatenates it with the second span of b (to get the second span of a). For tree-adjoining grammars, rules have the form:

$$a[x_1 y_1, y_2 x_2] \to b[x_1 x_2]\, c[y_1 y_2]$$

The fan-out of the grammar, f, is the maximum fan-out of its nonterminals:

$$f = \max_{a \in \mathcal{N}} \varphi(a). \tag{2}$$

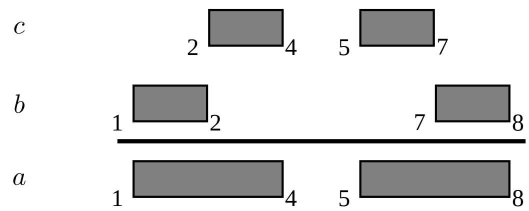

Figure 1
An example of a combination of spans for Tree Adjoining Grammars (TAG) for the adjunction operation in terms of binary LCFRS. The numbers denote the beginning (left) and end (right) endpoints for the spans. The figure denotes how two nonterminals B and C are combined together into a nonterminal A.

We sometimes refer to the *skeleton* of a grammar rule $a[\alpha] \rightarrow b[\beta]\, c[\gamma]$, which is just the context-free rule $a \rightarrow b\, c$, omitting the variables. In that context, a logical statement such as $a \rightarrow b\, c \in \mathcal{R}$ is true if there is any rule $a[\alpha] \rightarrow b[\beta]\, c[\gamma] \in \mathcal{R}$ with some α, β and γ.

By limiting ourselves to binary LCFRS grammars, we do not necessarily restrict the power of our results. Any LCFRS with arbitrary rank (i.e. with an arbitrary number of nonterminals in the right-hand side) can be converted to a binary LCFRS (with potentially a larger fan-out). See discussion in §7.5.

3. A Sketch of the Algorithm

Our algorithm for LCFRS string recognition is inspired by the algorithm of Valiant (1975). It introduces a few important novelties that make it possible to use matrix multiplication for the goal of LCFRS recognition.

The algorithm relies on the observation that it is possible to construct a matrix T with a specific non-associative multiplication and addition operator such that multiplying T by itself k times on the left or on the right yields k-step derivations for a given string. The row and column indices of the matrix together assemble a set of spans in the string (the fan-out of the grammar determines the number of spans). Each cell in the matrix keeps track, among other things, of the nonterminals that can dominate these spans. Therefore, computing the transitive closure of this matrix yields in each matrix cell the set of nonterminals that can dominate the assembled indices' spans for the specific string at hand.

There are several key differences between Valiant's algorithm and our algorithm. Valiant's algorithm has a rather simple matrix indexing scheme for the matrix: the rows correspond to the left endpoints of a span and the columns correspond to its right endpoints. Our matrix indexing scheme can mix both left endpoints and right endpoints at either the rows or the columns. This is necessary because with LCFRS, spans for the right-hand side of an LCFRS rule can combine in various ways into a new set of spans for the left-hand side.

In addition, our indexing scheme is "over-complete." This means that different cells in the matrix T (or its matrix powers) are equivalent and should consist of the same nonterminals. The reason we need such an over-complete scheme is again because of the possible ways spans of a right-hand side can combine in an LCFRS. To address this over-completeness, we introduce into the multiplication operator a "copy operation" that copies nonterminals between cells in order to maintain the same set of nonterminals in equivalent cells.

To give a preliminary example, consider the tree adjoining grammar rule shown in Figure 1. With our algorithm, this operation will translate into the following sequence of matrix transformations. We will start with the following matrices, T_1 and T_2:

$$\begin{array}{cc} T_1 & (2,7) \\ (1,8) & \left(\begin{array}{ccc} & \{\ldots,B,\ldots\} & \\ & & \\ & & \end{array}\right) \end{array} \qquad \begin{array}{cc} T_2 & (4,5) \\ (2,7) & \left(\begin{array}{ccc} & & \\ & \{\ldots,C,\ldots\} & \\ & & \end{array}\right) \end{array}.$$

For T_1, for example, the fact that B appears for the pair of addresses $(1,8)$ (for row) and $(2,7)$ for column denotes that B spans the constituents $(1,2)$ and $(7,8)$ in the string (this is assumed to be true – in practice, it is the result of a previous step of matrix multiplication). Similarly, with T_2, C spans the constituents $(2,4)$ and $(5,7)$.

The result of multiplying T_1 by T_2 is the following:

$$\begin{array}{cc} T_1T_2 & (4,5) \\ (1,8) & \left(\begin{array}{ccc} & \{\ldots,A,\ldots\} & \\ & & \\ & & \end{array}\right) \end{array}.$$

Now A appears in the cell that corresponds to the spans $(1,4)$ and $(5,8)$. This is the result of merging the spans $(1,2)$ with $(2,4)$ (left span of B and right span of C) into $(1,4)$ and the merging of the spans $(5,7)$ and $(7,8)$ (left span of C and right span of B) into $(5,8)$. Finally, an additional copying operation will lead to the following matrix:

$$\begin{array}{cc} T_3 & (5,8) \\ (1,4) & \left(\begin{array}{ccc} & \{\ldots,A,\ldots\} & \\ & & \\ & & \end{array}\right) \end{array}.$$

Here, we copy the nonterminal A from the address with the row $(1,8)$ and column $(4,5)$ into the address with the row $(1,4)$ and column $(5,8)$. Both of these addresses correspond to the same spans $(1,4)$ and $(5,8)$. Note that matrix row and column addresses can mix both starting points of spans and ending points of spans.

4. A Matrix Multiplication Algorithm for LCFRS

We turn next to give a description of the algorithm. Our description is constructed as following:

- In §4.1 we describe the basic matrix structure which is used for LCFRS recognition. This construction depends on a parameter d, the contact rank, which is a function of the underlying LCFRS grammar we parse with. We also describe how to create a seed matrix, for which we need to compute the transitive closure.
- In §4.2 we define the multiplication operator between cells of the matrices we use. This multiplication operator is distributive, but not associative, and as such, we use Valiant's specialized transitive closure algorithm to compute transitive closure of the seed matrix given a string.
- In §4.3 we define the d parameter. The smaller d is, the more efficient it is to parse with the specific grammar.
- In §4.4 we define when a binary LCFRS is "balanced." This is an end case that increases the final complexity of our algorithm by a factor of $O(n)$. Nevertheless, it is an important end case that appears in applications, such as inversion transduction grammars.
- In §4.5 we tie things up, and show that computing the transitive closure of the seed matrix we define in §4.1 yields a recognition algorithm for LCFRS.

4.1 Matrix Structure

The algorithm will seek to compute the transitive closure of a seed matrix $T(d)$, where d is a constant determined by the grammar (see §4.3). The matrix rows and columns are indexed by the set $N(d)$ defined as:

$$N(d) = \bigcup_{d'=1}^{d} [n]_0^{d'}, \tag{3}$$

where n denotes the length of the sentence. If $i \in N(d)$, then we define $\alpha(i)$ to be the set of all elements that appear in the sequence i. For $i, j \in N(d)$, we define $m(i,j)$ to be the set of $f' = \frac{1}{2}|\alpha(i) \cup \alpha(j)|$ pairs $\{(\ell_1, \ell_2), (\ell_3, \ell_4), \ldots, (\ell_{2f'-1}, \ell_{2f'})\}$ such that $\ell_k < \ell_{k+1}$ for $k \in [2f'-1]$ and $\ell_k \in \alpha(i) \cup \alpha(j)$ for $k \in [2f']$. (Whenever $\min \alpha(j) \leq \min \alpha(i)$, we define $m(i,j) = \perp$. The interpretation of this is that ℓ_1 should always belong to $\alpha(i)$ and not $\alpha(j)$. See more details in §4.2. In addition, if $|\alpha(i) \cup \alpha(j)|$ is not an even number, we also set $m(i,j) = \perp$.) This means that $m(i,j)$ takes as input the two sequences in matrix indices, merges them, sorts them, then divides this sorted list into a set of f' consecutive pairs.

We define a partial order on elements of $N(d)$. For any $i, j \in N$ ($i \neq j$) such that $m(i,j) \neq \perp$, we say that $i < j$ if $\min \alpha(i) < \min \alpha(j)$. We assume that the rows and columns of our matrices are ordered in some order consistent with this partial order. For the rest of the discussion, we assume that d is a constant, and refer to $T(d)$ as T and $N(d)$ as N.

Inputs: An LCFRS grammar as defined in §2 and a sentence $w_1 \cdots w_n$.

Outputs: A seed matrix T with rows and columns indexed by N, such that each cell in T is a subset of M

Algorithm:

- Set $T_{ij} = \emptyset$ for all $i, j \in N$.
- For each $i, j \in N$ such that $i < j$
  - $T_{ij} \leftarrow T_{ij} \cup \{\mathrm{ToCol}\}$ if $j = \mathrm{insert}(i, x)$ for some x
  - $T_{ij} \leftarrow T_{ij} \cup \{\mathrm{FromRow}\}$ if $i = \mathrm{remove}(j, x)$ for some x
  - $T_{ij} \leftarrow T_{ij} \cup \{\mathrm{ToRow}\}$ if $i = \mathrm{insert}(j, x)$ for some x
  - $T_{ij} \leftarrow T_{ij} \cup \{\mathrm{FromCol}\}$ if $j = \mathrm{remove}(i, x)$ for some x
- For each $i, j \in N$, for each nonterminal $a \in \mathcal{N}$, set $T_{ij} \leftarrow T_{ij} \cup \{a\}$ if $m(i, j) = \{(\ell_1, \ell_2), (\ell_3, \ell_4), \ldots, (\ell_{2\varphi(a)-1}, \ell_{2\varphi(a)})\}$ and:
  - For every pair $(\ell, \ell') \in m(i, j)$ it holds that $\ell' = \ell + 1$.
  - There is a rule in the grammar $a[\alpha] \rightarrow x_1[\beta_1], \ldots, x_f[\beta_f]$ such that $\alpha_k = \beta_{k'}$ for some $k' \in [f]$, and $w_{\ell_{2k}} = x_{k'}$ (i.e. $x_{k'}$ is the ℓ_{2k} word in the sentence).

Figure 2
An algorithm for computing the seed matrix T. The function $\mathrm{remove}(v, x)$ takes a sequence of integers v and removes x from it, if it is in there. The function $\mathrm{insert}(v, x)$ takes a sequence of integers and adds x to it.

We also define the following set

$$M = \mathcal{N} \cup \{\mathrm{FromRow}, \mathrm{ToCol}, \mathrm{FromCol}, \mathrm{ToRow}\} \tag{4}$$

where FromRow, ToCol, FromCol, and ToRow are four special pre-defined symbols. Each cell T_{ij} in T is such that $T_{ij} \subset M$.

The intuition behind matrices of the type of T (meaning, T, and as we see later, products of T with itself, or its transitive closure) is that each cell indexed by (i, j) in such a matrix consists of all nonterminals that can be generated by the grammar when parsing a sentence such that these nonterminals span the constituents $m(i, j)$ (whenever $m(i, j) \neq \bot$). The additional FromRow, ToCol, FromCol and ToRow symbols are symbols that indicate to the matrix multiplication operator that a "copying operation" should happen between equivalent cells (§4.2).

Figure 2 gives an algorithm to seed the matrix T with elements, corresponding to chart elements for "preterminal rules." The matrix is upper-triangular if we use the order $<$ over the indices of the matrix, as defined above. See more about upper triangularity in §4.2.

4.2 Definition of Multiplication Operator

We need to define a multiplication operator $\otimes$ between a pair of elements $R, S \subset M$. Such a multiplication operator induces multiplication between matrices of the type of T, just by defining for two such matrices, T_1 and T_2, a new matrix of the same size $T_1 \otimes T_2$ such that:

$$[T_1 \otimes T_2]_{ij} = \bigcup_{k \in N} \left([T_1]_{ik} \otimes [T_2]_{kj} \right), \tag{5}$$

We also the $\cup$ symbol is used to denote coordinate-wise union of cells in the matrices it operates on.

The operator $\otimes$ we define is not associative, but it is distributive over $\cup$. This means that for $R, S_1, S_2 \subset M$ it holds that:

$$R \otimes (S_1 \cup S_2) = (R \otimes S_1) \cup (R \otimes S_1). \tag{6}$$

In addition, whenever $R = \emptyset$, then for any S, $R \otimes S = S \otimes R = \emptyset$. This property maintains the upper-triangularity of the transitive closure of T.

Figure 3 gives the algorithm for multiplying two elements of the matrix. The algorithm is composed of two components. The first component (the second bullet point in Figure 3) adds nonterminals, for example, a, to cell (i, j), if there is some b and c in (i, k) and (k, j), respectively, such that there exists a rule $a \to b\, c$ *and* the span points denoted by i and j denote together the merge of the spans in (i, k) and (k, j), where k are the contact endpoints concatenating i and j.

In order to make this first component valid, we have to make sure that k can indeed serve as a concatenation point for (i, j). This is done by ensuring that the Non-overlapping Interval Condition and the Linked Sequence Pair Condition are satisfied. These two conditions can be verified in $O(1)$ time, independently of the string length n.

The only issue with the first component of the algorithm is that it is sound, but not complete. This means that if we were to use just this component in the algorithm, then we would get in each cell (i, j) of the transitive closure of T a *subset* of the possible nonterminals that can span $m(i, j)$. The reason this happens is that our addressing scheme is "over-complete." This means that any pair of addresses (i, j) and (k, ℓ) are equivalent if $m(i, j) = m(k, \ell)$.

This means that we need to ensure that the transitive closure, using $\otimes$, propagates, or *copies*, nonterminals from one cell to its equivalents. This is done by the second component of the algorithm, in bullet points 3–6. The algorithm does this kind of copying by using a set of four special "copy" symbols, $\{\mathrm{FromCol}, \mathrm{FromRow}, \mathrm{ToCol}, \mathrm{ToRow}\}$. These symbols copy nonterminals from one cell to the other in multiple stages.

Suppose that we need to copy a nonterminal from cell (i, j) to cell (k, ℓ), where $m(i, j) = m(k, \ell)$, indicating that the two cells describe the same set of indices in the input string. We must move the indices in $\alpha(i) \cap \alpha(\ell)$ from the row address to the column address, and we must move the indices in $\alpha(j) \cap \alpha(k)$ from the column address to the row address. We will move one index at a time, adding nonterminals to intermediate cells along the way.

We now illustrate how our operations move a single index from a row address to a column address (moving from column to row is similar). Let x indicate the index we wish to move, meaning that we wish to copy a nonterminal in cell (i, j) to cell $(\mathrm{remove}(i, x), \mathrm{insert}(j, x))$. Because we want our overall parsing algorithm to take advantage of fast matrix multiplication, we accomplish the copy operations through a sequence of two matrix multiplications. Intuitively, because a single multiplication on the right affects only the column address, and a single matrix multiplication on the left affects only the row address of the cells, we must multiply twice to change both addresses. The first multiplication involves the nonterminal a in cell (i, j) in the left matrix, and a ToCol symbol in cell $(j, \mathrm{insert}(j, x))$ in the right matrix, resulting in a matrix with nonterminal a in cell $(i, \mathrm{insert}(j, x))$. This intermediate result is redundant in the sense that index x appears in both the row and column addresses. To remove x from the row address,

Inputs: A pair of elements $R, S \subset M$.

Outputs: A new subset of M, denoted by $(R \otimes S)$.

Algorithm:

- $(R \otimes S) = \emptyset$.
- For each pair of elements $r = b$ and $s = c$ where $r \in R$ and $s \in S$ add an element a to $(R \otimes S)$ if:
  - There is a binary rule in the LCFRS grammar such that it has the form $a[\alpha] \rightarrow b[\beta]\, c[\gamma]$.
  - Let $f_a = \frac{1}{2}\left(|\alpha(i) \cup \alpha(j)|\right)$. Continue only if $f_a = \varphi(a)$.
  - Let $f_b = \frac{1}{2}\left(|\alpha(i) \cup \alpha(k)|\right)$. (Note that $f_b = \varphi(b)$.)
  - Let $f_c = \frac{1}{2}\left(|\alpha(k) \cup \alpha(j)|\right)$. (Note that $f_c = \varphi(c)$.)
  - (**Non-overlapping Interval Condition**) Assume $m(i,k) = \{(\ell_1, \ell_2), \ldots, (\ell_{2f-1}, \ell_{2f_b})\}$ and $m(k,j) = \{(\ell'_1, \ell'_2), \ldots, (\ell'_{2f-1}, \ell'_{2f_c})\}$ then any pair $\mathcal{I}_0 \in m(i,k)$ and $\mathcal{I}_1 \in m(k,j)$ is such that $\mathcal{I}_0 \cap \mathcal{I}_1 = \emptyset$ where the notation of (ℓ, ℓ') is overloaded to denote the open interval of real numbers between ℓ and ℓ'.
  - (**Linked Sequence Pair Condition**) For the rule $a[\alpha] \rightarrow b[\beta]\, c[\gamma]$ above consider the following. Let $\alpha_g \in \mathcal{V}^+$ for $g \in [\varphi(a)]$ such that $\alpha_g = y_1 \cdots y_p$ (i.e. α_g is a sequence of p variables). Each variable in α_g is matched with a single variable in β or γ (by definition of LCFRS rule). For $p' \in [p]$, if $y_{p'} = \beta_q$ for some $q \in [\varphi(b)]$, then define $\pi(p')$ to be the pair (ℓ_{2q-1}, ℓ_{2q}). If $y_{p'} = \gamma_{q'}$ for some $q' \in [\varphi(c)]$, then define $\pi(p')$ to be $(\ell'_{2q'-1}, \ell_{2q'})$. Then, it must hold that $\pi(1), \ldots, \pi(p)$ is a linked sequence of pairs.
- For each pair of elements $r = a$ and $s = \mathrm{ToCol}$, add the element a to $(R \otimes S)$ if: $|\alpha(i) \cap \alpha(j)| = 1$.
- For each pair of elements $r = \mathrm{FromRow}$ and $s = a$, add the element a to $(R \otimes S)$ if: $\alpha(i) \cap \alpha(j) = \emptyset$ and $|\alpha(i) \cup \alpha(j)| = 2\varphi(a)$
- For each pair of elements $r = \mathrm{ToRow}$ and $s = a$, add the element a to $(R \otimes S)$ if: $|\alpha(i) \cap \alpha(j)| = 1$.
- For each pair of elements $r = a$ and $s = \mathrm{FromCol}$, add the element a to $(R \otimes S)$ if: $\alpha(i) \cap \alpha(j) = \emptyset$ and $|\alpha(i) \cup \alpha(j)| = 2\varphi(a)$

Figure 3
An algorithm for the product of two matrix elements. The function $\mathrm{setdiff}(A, B)$ for two sets A and B returns $(A \setminus B)$.

we multiply on the left with a matrix containing the symbol $\mathrm{FromRow}$ in cell $(\mathrm{remove}(i,x),i)$, resulting in a matrix with nonterminal a in cell $(\mathrm{remove}(i,x),\mathrm{insert}(j,x))$.

In order to guarantee that our operations copy nonterminals only into cells with equivalent addresses, the seed matrix contains the special symbol ToCol only in cells (j,k) such that $k = \mathrm{insert}(j,x)$ for some x. This guarantees that the ToCol operation only adds one index at a time to the column address. Furthermore, when ToCol in cell (j,k) combines with a nonterminal a in cell (i,j), the result contains a only if $|\alpha(i)\cap\alpha(k)| = 1$, guaranteeing that the index added to the column address was originally present in the row address.

Similar conditions apply to the $\mathrm{FromRow}$ operation. The seed matrix contains $\mathrm{FromRow}$ only in cells (i,j) such that $i = \mathrm{remove}(j,x)$ for some x, guaranteeing that the operation only removes one index at a time. Furthermore, when $\mathrm{FromRow}$ in cell (i,j) combines with a nonterminal a in cell (j,k), the result contains a only if $\alpha(i)\cap\alpha(k) = \emptyset$ and $|\alpha(i)\cup\alpha(k)| = 2\varphi(a)$. This guarantees that the new entry contains no index more than once and includes all the original indices, meaning that any index we remove from the row address is still present in the column address. Taken together, these conditions ensure that after a sequence of one ToCol and one ToRow, the new cells that a is copied into have the form $(\mathrm{remove}(i,x),\mathrm{insert}(j,x))$ for some x.

To move an index from the column address to the row address, we use one ToRow operation followed by one $\mathrm{FromCol}$ operation. The conditions on these two special symbols are analogous to the conditions on ToCol and $\mathrm{FromRow}$ outlined above, and ensure that we copy from cell (i,j) to cells of the form $(\mathrm{insert}(i,x),\mathrm{remove}(j,x))$ for some x.

Putting together sequences of these operations to move indices, we get the following lemma:

Lemma 1
Let (i,j) and (k,ℓ) be matrix addresses such that $m(i,j) = m(k,\ell)$, in a matrix T that is large enough that $d > \min\{|\alpha(i)|,|\alpha(j)|\}$ and $d > \min\{|\alpha(k)|,|\alpha(\ell)|\}$. Then, for any nonterminal a in cell (i,j) in T, a will also appear in cell (k,ℓ) of the power matrix $T^{(4d)}$.

Proof
Nonterminal a can be copied through a series of intermediate cells by moving one index at a time from i to ℓ, and from j to k. We begin by moving indices from either the row address i to the column address if $|\alpha(i)| > |\alpha(j)|$, or from the column address j to the row address otherwise. The condition on the size of d guarantees that we can form row and column addresses long enough to hold the redundant representations with one address shared between row and column. We must move up to d indices from row to column, and d indices from column to row. Each move takes two matrix multiplications, for a total of $4d$ matrix multiplications. ■

Upper-triangularity of T and products of T. In §4.1, we define the order by which the indices of the matrix are set. More specifically, i appears before j in this ordering scheme if $\min\alpha(i) < \min\alpha(j)$. Keeping in mind that the nonterminals in a cell (i,j) for T (or its products) are always a subset of the possible nonterminals that could span $m(i,j)$, this ensures that T and any of its products is upper triangular according to this order.

To see this, consider that we assume that if $\min\alpha(i) \geq \min\alpha(j)$, then $m(i,j) = \perp$. As such, T will be the empty set for all such pairs of i and j. All elements below the diagonal of T correspond to such elements.

Given that T is initialized to be upper-triangular, the properties of matrix multiplication guarantee that all matrix powers of T are upper-triangular. In terms of the grammar, when applying a rule $a \rightarrow b\,c$, our normal form for LCFRS rules ensures that the leftmost endpoint of b forms the leftmost endpoint of a. The nonterminal b appears in the left matrix, and c appears in the right matrix of the matrix multiplication that produces a. In the product matrix, a appears in a cell whose row address is the row address of b. This row address includes the leftmost endpoint of b, which is also the leftmost endpoint of a. The operations that copy indices also maintain upper-triangularity by never copying the first index of the row address; this is guaranteed by the condition the $i < j$ in the initialization of T.

4.3 Determining the Contact Rank

The dimensions of the matrix T (and its transitive closure) are $|N| \times |N|$. The set N is of size $O(n^d)$, where d is a function of the grammar. When a given pair of cells in two matrices of the type of T are multiplied, we are essentially combining endpoints from the first multiplicand column address with endpoints from the second multiplicand row address. As such, we have to ensure that d allows us to generate all possible sequences of endpoints that could potentially combine with a given fixed LCFRS.

We refer to the endpoints at which a rule's r.h.s. nonterminals meet as **combining points**. For example, in the simple case of a CFG with a rule $\mathrm{S} \rightarrow \mathrm{NP}\,\mathrm{VP}$, there is one combining point where NP and VP meet. For each rule r in the LCFRS grammar, we must be able to access the combining points as row and column addresses in order to apply the rule with matrix multiplication. Thus, d must be at least the maximum number of combining points of any rule in the grammar. The number of combining points $\delta(r)$ for a rule r can be computed by comparing the number of spans on the l.h.s. and r.h.s. of the rule:

$$\delta(a[\alpha] \rightarrow b[\beta]\, c[\gamma]) = \varphi(c) + \varphi(b) - \varphi(a). \tag{7}$$

Note that $\delta(r)$ depends only on the skeleton of r (see §2), and therefore it can be denoted by $\delta(a \rightarrow b\, c)$.

For each nonterminal on the r.h.s. of the rule, the address of its matrix cell consists of the combination points in one dimension (either row or column), and the other points in the other dimension of the matrix. For r.h.s. nonterminal b in rule $a \rightarrow b\, c$, the number of non-combination endpoints is:

$$2\varphi(b) - \delta(a \rightarrow b\, c). \tag{8}$$

Thus, taking the maximum size over all addresses in the grammar, the largest addresses needed are of length:

$$d = \max_{a \rightarrow b\, c \in \mathcal{R}} \max \left\{ \begin{array}{c} \delta(a \rightarrow b\, c), \\ 2\varphi(b) - \delta(a \rightarrow b\, c), \\ 2\varphi(c) - \delta(a \rightarrow b\, c) \end{array} \right\}. \tag{9}$$

We call this number the *contact rank* of the grammar. A simple algebraic manipulation shows that the contact rank can be expressed as follows:

$$d = \max_{a \rightarrow b\, c \in \mathcal{R}} \max \left\{ \begin{array}{c} \varphi(a) + \varphi(b) - \varphi(c), \\ \varphi(a) - \varphi(b) + \varphi(c), \\ -\varphi(a) + \varphi(b) + \varphi(c) \end{array} \right\}. \tag{10}$$

4.4 Balanced Grammars

Our matrix representation requires that a nonterminal appears in more than one equivalent cell in the matrix, and the specific set of cells required depends on the specific patterns in which spans are combined in the LCFRS grammar. We now present a precise description of these cells by defining the **configuration** of a nonterminal in a rule.

For each of the three nonterminals involved in a rule, the configuration is the set of endpoints in the row address of the nonterminal's matrix cell. To make this precise, for a nonterminal b with fan-out $\varphi(b)$, we number the endpoints of spans with integers in the range 1 to $2\varphi(a)$. In a rule $a[\alpha] \rightarrow b[\beta]\, c[\gamma]$, the configuration of b is the subset C of $[2\varphi(b)]$ of endpoints of b that combine with endpoints of c in order to form a single span of a. Formally, let $\beta = \beta_1 \cdots \beta_{\varphi(b)}$, and let $\alpha = \langle \alpha_{1,1} \cdots \alpha_{1,n_1}, \ldots, \alpha_{\varphi(a),1} \cdots \alpha_{\varphi(a),n_{\varphi(a)}} \rangle$. Then

$$C_2(r) = \{2i : \beta_i = \alpha_{j,n_j} \text{ for some } j\} \cup \{2i-1 : \beta_i = \alpha_{j,1} \text{ for some } j\}$$

where the first set defines right ends of spans of b that are right ends of some span of a, and the second set defines lefts ends of spans of b that are left ends of some span of a. Similarly, for the second r.h.s. nonterminal of a rule r,

$$C_3(r) = \{2i : \gamma_i = \alpha_{j,k} \text{ for some } 1 \leq k < n_j\} \cup \{2i-1 : \gamma_i = \alpha_{j,k} \text{ for some } 1 < k \leq n_j\}$$

where the first set defines right ends of spans of c that are internal to some span of a, and the second set defines lefts ends of spans of c that are internal to some span of a. For the l.h.s. nonterminal a of the rule, matrix multiplication will produce an entry in the matrix cell where the row address corresponds to the endpoints from b, and the column address corresponds to the endpoints from c. To capture this partition of the endpoints of a, we define

$$C_1(r) = \{2i : \alpha_{i,n_i} = \beta_j \text{ for some } j\} \cup \{2i-1 : \alpha_{i,1} = \beta_j \text{ for some } j\},$$

where the first set defines right ends of spans of a that are formed from b, and the second set defines left ends of spans of a that are formed from b.

To apply a rule $r : a[\alpha] \rightarrow b[\beta]\, c[\gamma]$, we must have an entry for b in cell $(C_2(r), [2\varphi(b)] - C_2(r))$, and an entry for c in cell $(C_3(r), [2\varphi(c)] - C_3(r))$.

We define the **configuration set** of a nonterminal a to the the set of all configurations in which a appears in a grammar rule, including both appearances in the r.h.s. and as the l.h.s.

$$C(a) = \left(\bigcup_{r:lhs(r)=a} \{C_1(r)\} \right) \cup \left(\bigcup_{r:rhs1(r)=a} \{C_2(r)\} \right) \cup \left(\bigcup_{r:rhs2(r)=a} \{C_3(r)\} \right)$$

A configuration C of nonterminal b is **balanced** if $|C| = \varphi(b)$. This means that the number of contact points and non-contact points are the same.

The contact rank d defined in the previous section is the maximum size of any configuration of any nonterminal in any rule. For a given nonterminal b, if $\varphi(b) < d$, then we can copy entries between equivalent cells. To see this, suppose that we are moving from cell (i, j) to (k, ℓ) where the length of i is greater than the length of j. As long as we move the first index from row to column, rather than from column to row, the intermediate results will require addresses no longer than the length of i.

However, if $\varphi(b) = d$, then every configuration in which b appears is balanced:

$$\forall C \in C(b) \quad |C| = \varphi(b)$$

If $\varphi(b) = d$ and b appears in more than one configuration, that is, $|C(b)| > 1$, it is impossible to copy entries for b between the cells using a matrix of size n^d. This is because we cannot move indices from row to column or from column to row without creating an intermediate row or column address of length greater than d as a result of the first ToCol or ToRow operation.

We define a **balanced grammar** to be a grammar containing a nonterminal b such that $\varphi(b) = d$, and $|C(b)| > 1$. The following condition will determine which of two alternative methods we use for the top level of our parsing algorithm.

Condition 4.1
Unbalanced Grammar Condition There is no nonterminal b such that $\varphi(b) = d$ and $|C(b)| > 1$.

This condition guarantees that we can move nonterminals as necessary with matrix multiplication:

Lemma 2
Let (i, j) and (k, ℓ) be matrix addresses such that $m(i, j) = m(k, \ell)$. Under Condition 4.1, for any nonterminal a in cell (i, j) in T, a will also appear in cell (k, ℓ) of the power matrix $T^{(4d)}$.

Proof
The number of a's endpoints is $2\varphi(a) = |\alpha(i)| + |\alpha(j)| = |\alpha(k)| + |\alpha(\ell)|$. If the grammar is not balanced, then $d > \varphi(a)$, and therefore $d > \min\{|\alpha(i), \alpha(j)\}$ and $d > \min\{|\alpha(k)|, |\alpha(\ell)|\}$. By Lemma 1, a will appear in cell (k, ℓ) of the power matrix $T^{(4d)}$. ■

4.5 Computing the Transitive Closure of T

The transitive closure of T is defined based on the matrix multiplication operator described in Eq. 5. With T being the seed matrix, we define

$$T^+ = T^{(1)} \cup T^{(2)} \cup \cdots, \tag{11}$$

where $T^{(i)}$ is defined recursively as:

$$T^{(1)} = T \tag{12}$$

$$T^{(i)} = \bigcup_{j=1}^{i-1} \left(T^{(j)} \otimes T^{(i-j)} \right). \tag{13}$$

Under Condition 4.1, one can show that given an LCFRS derivation tree t over the input string, each node in t must appear in the transitive closure matrix T^+. Specifically, for each node in t representing nonterminal a spanning endpoints $\{(\ell_1, \ell_2), (\ell_3, \ell_4), \ldots, (\ell_{2\varphi(a)-1}, \ell_{2\varphi(a)})\}$, at each cell $T^+_{i,j}$ in the matrix such that $m(i, j) = \{(\ell_1, \ell_2), (\ell_3, \ell_4), \ldots, (\ell_{2\varphi(a)-1}, \ell_{2\varphi(a)})\}$, contains a. This can be shown by induction over the length of the LCFRS derivations. Derivations consisting of a single rule $a[\alpha] \to b[\beta]\, c[\gamma]$ produce $a \in T^{(2)}$ for i and j corresponding the non-combination points of b and c. For all other i and j such that $m(i, j) =$

$\{(\ell_1, \ell_2), (\ell_3, \ell_4), \ldots, (\ell_{2\varphi(a)-1}, \ell_{2\varphi(a)})\}$, an entry is produced in $T^{(4d)}_{ij}$ by Lemma 2. By induction, $T^{s(4d+2)}$ contains entries for all LCFRS derivations of depth s, and T^{+} contains entries for all LCFRS derivations of any length.

In the other direction, we need to show that all entries a in T^{+} correspond to a valid LCFRS derivation of nonterminal a spanning endpoints $m(i, j)$. This can be shown by induction over the number of matrix multiplications. During each multiplication, entries created in the product matrix correspond either to the application of an LCFRS rule with l.h.s. a, or to the movement of an index between row and column address for a previously recognized instance of a. This leads to the following result:

Theorem 1
Under Condition 4.1, the transitive closure of T is such that $[T^{+}]_{ij}$ represents the set of nonterminals that are derivable for the given spans in $m(i, j)$.

The transitive closure still yields a useful result, even when Condition 4.1 does not hold. To show how it is useful, we need to define the "copying" operator, Π, which takes a matrix A of the same type of T, and sets $\Pi(A)$ using the following procedure:

- Define $e(i, j) = \{(i', j') \mid m(i', j') = m(i, j)\}$, i.e. the set of equivalent configurations to (i, j).
- Set $[\Pi(A)]_{ij} = \bigcup_{(i',j') \in e(i,j)} A_{i'j'}$.

This means that Π takes a completion step, and copies all nonterminals between all equivalent addresses in A. In that case, we have the following result:

Theorem 2
The transitive closure of T is such that $[T^{+}]_{ij}$ represents a *subset* of nonterminals that are derivable for the given spans in $m(i, j)$. In addition, if $(\Pi(T))^{+}$ is computed, then either it equals T^{+}, or at least one nonterminal is added to one of the cells (i, j) such that this nonterminal is derivable for the corresponding spans $m(i, j)$.

Note that the Π operator can be implemented such that it operates in time $O(n^d)$. All it requires is just taking $O(n^d)$ unions of sets (corresponding to the sets of nonterminals in the matrix cells), where each set is of size $O(1)$ with respect to the sentence length (i.e. it is constant with respect to the grammar).

Theorem 2 leads to a recognition algorithm for binary LCFRS that do not satisfy Condition 4.1 (we also assume that these binary LCFRS would not have unary cycles or ϵ rules). This algorithm is given in Figure 5. It operates by iterating through transitive closure step and copying steps until convergence. When we take the transitive closure of T, we are essentially computing a subset of the derivable nonterminals. Then, the copying step (with Π) propagates nonterminals through equivalent cells. Now, if we take the transitive closure again, and there is any way to derive new nonterminals because of the copying step, the resulting matrix will have at least one new nonterminal. Otherwise, it will not change, and as such, we recognized all possible derivable nonterminals in each cell.

Reduction of Transitive Closure to Boolean Matrix Multiplication. Valiant showed that his algorithm for computing the multiplication of two matrices, in terms of multiplication operator similar to ours, can be reduced to the problem of Boolean matrix multiplication. His transitive closure algorithm requires as a black box this two-matrix multiplication algorithm.

We follow here a similar argument. We can use Valiant's algorithm for the computation of the transitive closure, since our multiplication operator is distributive (with respect to $\cup$). To complete our argument, we need to show, similarly to Valiant, that the product of two matrices using our multiplication operator can be reduced to Boolean matrix multiplication.

Consider the problem of multiplication a matrix T_1 and T_2, and say $T_1 \otimes T_2 = T_3$. To reduce it to Boolean matrix multiplication, we create $2|\mathcal{N}|$ pairs of matrices, G_a and H_a, where a ranges over $\mathcal{N}$. The size of G_a and H_a is $N \times N$. For $a \in \mathcal{N}$, we set $[G_a]_{ij}$ to be 1 if the nonterminal a appears in $[T_1]_{ij}$, and similarly, we set $[H_a]_{ij}$ to be 1 if the nonterminal a appears in $[T_2]_{ij}$. All other cells, in both G_a and H_a, are set to 0. Note that G_a and H_a for all $a \in \mathcal{N}$ are upper triangular Boolean matrices.

In addition, we create 4 additional matrices, for each element in the set $\{\mathrm{FromCol}, \mathrm{FromRow}, \mathrm{ToCol}, \mathrm{ToRow}\}$. The first matrix is G_{FromRow}, for which $[G_{\mathrm{FromRow}}]_{ij} = 1$ only in cells where $i = \mathrm{remove}(j, x)$ for some $x \in [n]_0$. The second matrix is H_{ToCol}, for which $[H_{\mathrm{ToCol}}]_{ij} = 1$ only in cells where $j = \mathrm{insert}(i, x)$ for some $x \in [n]_0$. The third matrix is G_{ToRow}, for which $[G_{\mathrm{ToRow}}]_{ij} = 1$ only in cells where $i = \mathrm{insert}(j, x)$ for some $x \in [n]_0$. The fourth matrix is H_{FromCol}, for which $[H_{\mathrm{FromCol}}]_{ij} = 1$ only in cells where $j = \mathrm{remove}(i, x)$ for some $x \in [n]_0$.

Now, for each pair $(b, c) \in \mathcal{N} \times \mathcal{N}$, we compute the matrix $I_{bc} = G_b H_c$. The total number of matrix multiplications required for that is constant in n, it is $|\mathcal{N}|^2$. Now, T_3 can be obtained in the following way:

- For each $a \in \mathcal{N}$, for each rule $a \to b\,c$, check whether $[I_{bc}]_{ij} = 1$. If not, then do not add a to $[T_3]_{ij}$.
- If $[I_{bc}]_{ij} = 1$ for the rule above, add a to $[T_3]_{ij}$ if the Non-overlapping Interval Condition and Linked Sequence Pair Condition (in Figure 3) are satisfied.
- For each $a \in \mathcal{N}$, compute $J_a = G_a H_{\mathrm{ToCol}}$. For each (i, j), add a to $[T_3]_{ij}$ if $|\alpha(i) \cap \alpha(j)| = 1$ and $[J_a]_{ij} = 1$.
- For each $a \in \mathcal{N}$, compute $J_a = G_a H_{\mathrm{FromCol}}$. For each (i, j), add a to $[T_3]_{ij}$ if $\alpha(i) \cap \alpha(j) = \emptyset$ and $|\alpha(i) \cup \alpha(j)| = 2\varphi(a)$, and $[J_a]_{ij} = 1$.
- For each $a \in \mathcal{N}$, compute $J_a = G_{\mathrm{ToRow}} H_a$. For each (i, j), add a to $[T_3]_{ij}$ if $|\alpha(i) \cap \alpha(j)| = 1$ and $[J_a]_{ij} = 1$.
- For each $a \in \mathcal{N}$, compute $J_a = G_{\mathrm{FromRow}} H_a$. For each (i, j), add a to $[T_3]_{ij}$ if $\alpha(i) \cap \alpha(j) = \emptyset$ and $|\alpha(i) \cup \alpha(j)| = 2\varphi(a)$, and $[J_a]_{ij} = 1$.

The first two conditions above process grammars rules, adding the l.h.s. nonterminal if the two r.h.s. nonterminals have been seen in matching spans in the string.

The final four conditions above handle copying of nonterminals between cells in the matrix that specify equivalent spans in the string. The matrices G_{ToRow} and H_{ToCol} create entries in cells (i, j) such that i and j contain one index in common. These cells exist solely for the purposes of subsequent combination with matrices H_{FromCol} and G_{FromRow}, which remove the common index from the column and row address respectively. After this sequence of two multiplications, an entry in cell (i, j) of the matrix can also be found in cell $(\mathrm{remove}(i, x), \mathrm{insert}(j, x))$ for any x in the original row address i. Similarly, entries are copied to any cell $(\mathrm{insert}(i, x), \mathrm{remove}(j, x))$ for any x in j. After a sequence of $4d$ such

Inputs: An LCFRS grammar as defined in §2 that satisfies Condition 4.1 and a sentence $w_1 \cdots w_n$.

Outputs: True if $w_1 \cdots w_n$ is in the language of the grammar, False otherwise.

Algorithm:

- Compute T as the seed matrix using the algorithm in Figure 2.
- Compute the transitive closure of T with the multiplication operator in Figure 3 and using Boolean matrix multiplication (§4.5).
- Return True if S belongs to the cell $(0, n)$ in the computed transitive closure, and False otherwise.

Figure 4
Algorithm for recognizing binary linear context-free rewriting systems when Condition 4.1 is satisfied by the LCFRS.

moves of an individual index, we are guaranteed to have entries in all cells (i', j') such that $m(i', j') = m(i, j)$.

The final parsing algorithm is given in Figure 4. It works by computing the seed matrix T, and then finding its transitive closure. Finally, it checks whether the start symbol appears in a cell with an address that spans the whole string. If so, the string is in the language of the grammar.

5. Computational Complexity Analysis

As mentioned in the previous section, the algorithm in Figure 4 finds the transitive closure of a matrix under our definition of matrix multiplication. The operations $\cup$ and $\otimes$ used in our matrix multiplication distribute. The $\otimes$ operator takes the cross product of two sets, and applies a filtering condition to the results; the fact that $(x \otimes y) \cup (x \otimes z) = x \otimes (y \cup x)$ follows from the fact that it does not matter whether we take the cross product of the union, or the union of the cross product. However, unlike in the case of standard matrix multiplication, our $\otimes$ operation is not associative. In general, $x \otimes (y \otimes z) \neq (x \otimes y) \otimes z$, because the combination of y and z may be allowed by the LCFRS grammar, while the combination of x and y is not.

We can use the algorithm of Valiant for finding the closure of distributive, non-associative matrix multiplication. Valiant describes an algorithm with time complexity $M(m)$, where $M(m)$ is the complexity of Boolean matrix multiplication for $m \times m$ matrices. When Valiant's paper was published, the best well-known algorithm known for such multiplication was Strassen's algorithm, with $M(m) = O(n^{2.8704})$. Since then, it is known that $M(n) = O(n^{\omega})$ for $\omega < 2.38$ (see also §1). There are ongoing attempts to further reduce ω, or find lower bounds for $M(m)$.

With our algorithm, the size of multiplied matrices is $O(n^d)$, giving an overall complexity of $O(M(n^d)) = O(n^{\omega d})$. Parsing a binary LCFRS rule with standard chart parsing techniques requires time $O(n^{\varphi(a)+\varphi(b)+\varphi(c)})$.

Let $p = \max_{a \to b\, c \in \mathcal{R}} (\varphi(a) + \varphi(b) + \varphi(c))$. The worst-case complexity of LCFRS chart parsing techniques is $O(n^p)$. We can now ask the question: in which case the algorithm in Figure 4 is asymptotically more efficient than standard chart parsing techniques with respect to n? That is, in which cases is $n^{d\omega} = o(n^p)$?

Clearly, this would hold whenever $d\omega < p$. By definition of d and p, a sufficient condition for that is that for any rule $a \to b\,c \in \mathcal{R}$ it holds that:[2]

$$\max \left\{ \begin{array}{c} \varphi(a) + \varphi(b) - \varphi(c), \\ \varphi(a) - \varphi(b) + \varphi(c), \\ -\varphi(a) + \varphi(b) + \varphi(c) \end{array} \right\} < \frac{1}{\omega} \left(\varphi(a) + \varphi(b) + \varphi(c) \right). \tag{14}$$

This means that for any rule, the following conditions should hold:

$$\omega(\varphi(a) + \varphi(b) - \varphi(c)) < \varphi(a) + \varphi(b) + \varphi(c), \tag{15}$$

$$\omega(\varphi(a) - \varphi(b) + \varphi(c)) < \varphi(a) + \varphi(b) + \varphi(c), \tag{16}$$

$$\omega(-\varphi(a) + \varphi(b) + \varphi(c)) < \varphi(a) + \varphi(b) + \varphi(c). \tag{17}$$

Algebraic manipulation shows that this is equivalent to having:

$$\varphi(a) + \varphi(b) < \left(\frac{\omega + 1}{\omega - 1} \right) \varphi(c), \tag{18}$$

$$\varphi(b) + \varphi(c) < \left(\frac{\omega + 1}{\omega - 1} \right) \varphi(a), \tag{19}$$

$$\varphi(c) + \varphi(a) < \left(\frac{\omega + 1}{\omega - 1} \right) \varphi(b). \tag{20}$$

For the best well-known algorithm for matrix multiplication, it holds that:

$$\frac{\omega + 1}{\omega - 1} > 2.44. \tag{21}$$

For Strassen's algorithm, it holds that:

$$\frac{\omega + 1}{\omega - 1} > 2.06. \tag{22}$$

We turn now to analyze the complexity of the algorithm in Figure 5. It works by iteratively applying the transitive closure and the copying operator until convergence. Each transitive closure has the asymptotic complexity of $O(n^{\omega d})$. Each Π application has the asymptotic complexity of $O(n^d)$. As such, the total complexity is $O(tn^{\omega d})$, where t is the number of iterations required to converge. At each iteration, we discover at least one new nonterminal. The total number of nodes in the derivation for the recognized string is $O(n)$ (assuming no unary cycles or ϵ rules). As such $t = O(n)$, and the total complexity of this algorithm is $O(n^{\omega d + 1})$.

6. Applications

Our algorithm is a recognition algorithm which is applicable to binary LCFRS. As such, our algorithm can be applied to any LCFRS, by first reducing it to a binary LCFRS. We discuss

2 For two sets of real numbers, A and B, it holds that if for any $a \in A$ there is a $b \in B$ such that $a < b$, then $\max A < \max B$.

Inputs: An LCFRS grammar as defined in §2 and a sentence $w_1 \cdots w_n$.

Outputs: True if $w_1 \cdots w_n$ is in the language of the grammar, False otherwise.

Algorithm:

- Compute T as the seed matrix using the algorithm in Figure 2.
- Repeat until T does not change: $T \leftarrow (\Pi(T))^+$.
- Return True if S belongs to the cell $(0, n)$ in the computed transitive closure, and False otherwise.

Figure 5
Algorithm for recognizing binary LCFRS when Condition 4.1 is not necessarily satisfied by the LCFRS.

results for specific classes of LCFRS in this section, and return to the general binarization process in §7.5.

LCFRS subsumes context-free grammars, which was the formalism that Valiant (1975) focused on. Valiant showed that the problem of CFG recognition can be reduced to the problem of matrix multiplication, and as such, the complexity of CFG recognition in that case is $O(n^\omega)$. Our result reinforces Valiant's result. CFGs (in Chomsky normal form) can be reduced to a binary LCFRS with $f = 1$. As such, $d = 1$ for CFGs, and our algorithm yields a complexity of $O(n^\omega)$. (Note that CFGs satisfy Condition 4.1, and therefore we can use a single transitive closure step.)

LCFRS is a broad family of grammars, and it subsumes many other well-known grammar formalisms, some which were discovered or developed independently of LCFRS. Two such formalisms are tree adjoining grammars (Joshi and Schabes 1997) and synchronous context-free grammars. In the next two sections, we explain how our algorithmic result applies to these two formalisms.

6.1 Mildly Context-Sensitive Language Recognition

Linear context-free rewriting systems fall under the realm of mildly context-sensitive grammar formalisms. They subsume four important mildly context-sensitive formalisms that were developed independently and later shown to be weakly equivalent by Vijay-Shanker and Weir (1994): tree adjoining grammars (Joshi and Schabes 1997), linear indexed grammars (Gazdar 1988), head grammars (Pollard 1984) and combinatory categorial grammars (Steedman 2000). Weak equivalence here refers to the idea that any language generated by a grammar in one of these formalisms, can be also be generated by some grammar in any of the other formalisms among the four. It can be verified that all of these formalisms satisfy Condition 4.1, and as such, the algorithm in Figure 5 applies to them.

Rajasekaran and Yooseph (1998) showed that tree adjoining grammars can be parsed with an asymptotic complexity of $O(M(n^2)) = O(n^{4.76})$. While he did not discuss that, the weak equivalence between the four formalisms mentioned above implies that all of them can be parsed in time $O(M(n^2))$. Our algorithm reinforces this result. We now give the details.

Our starting point for this discussion is head grammars. Head grammars are a specific case of linear context-free rewriting systems, not just in the formal languages they define – but also in the way these grammars are described. They are described using *concatenation* production rules and *wrapping* production rules, which are directly transferable to LCFRS notation. Their fan-out is

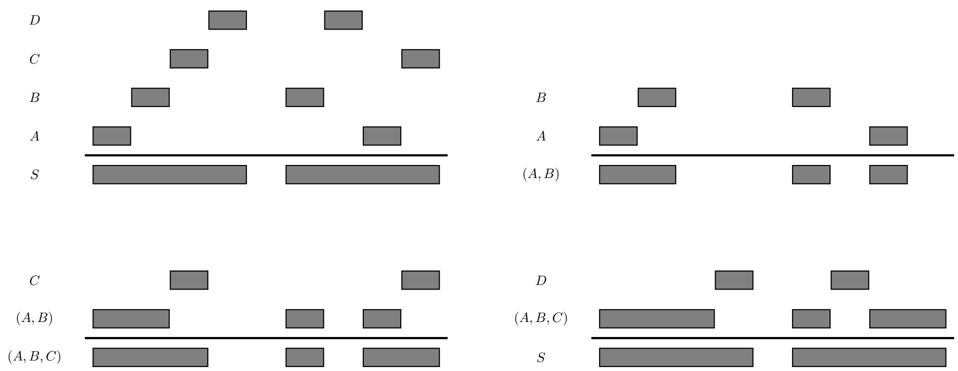

Figure 6
Upper left: Combination of spans for SCFG rule $S \rightarrow ABCD,\ BDAC$. Upper right and bottom row: three steps in parsing binarized rule.

2. We focus in this discussion on "binary head grammars," defined analogously to binary LCFRS – the rank of all production rules has to be 2. The contact rank of binary head grammars is 2. As such, our paper shows that the complexity of recognizing binary head grammar languages is $O(M(n^2)) = O(n^{4.76})$.

Vijay-Shanker and Weir (1994) show that *linear indexed grammars* (LIGs) can actually be reduced to binary head grammars. Linear indexed grammars are extensions of CFGs, a linguistically-motivated restricted version of indexed grammars, the latter of which were developed by Aho (1968) for the goal of handling variable binding in programming languages. The main difference between LIGs and CFGs is that the nonterminals carry a "stack," with a separate set of stack symbols. Production rules with LIGs copy the stack on the left-hand side to *one* of the nonterminal stacks in the righthand side,[3] with potentially pushing or popping one symbol in the new copy of the stack. For our discussion, the main important detail about the reduction of LIGs to head grammars is that it preserves the rank of the production rules. As such, our paper shows that binary LIGs can also be recognized in time $O(n^{4.76})$.

Vijay-Shanker and Weir (1994) additionally address the issue of reducing combinatory categorial grammars to LIGs. The combinators they allow are function application and function composition. The key detail here is that their reduction of CCG is to an LIG with rank 2, and as such, our algorithm applies to CCGs as well, which can be recognized in time $O(n^{4.76})$.

Finally, Vijay-Shanker and Weir (1994) reduced tree-adjoining grammars to combinatory categorial grammars. The TAGs they tackle are in "normal form," such that the auxiliary trees are binary (all TAGs can be reduced to normal form TAGs). Such TAGs can be converted to weakly equivalent CCG (but not necessarily strongly equivalent), and as such, our algorithm applies to TAGs as well. As mentioned above, this finding supports the finding of Rajasekaran and Yooseph (1998), who showed that TAG can be recognized in time $O(M(n^2))$.

For an earlier discussion connections between TAG parsing and Boolean matrix multiplication, see Satta (1994).

3 General indexed grammars copy the stack to multiple nonterminals on the right-hand side.

6.2 Synchronous context-free grammars

Synchronous Context-Free Grammars (SCFGs) are widely used in machine translation to model the simultaneous derivation of translationally equivalent strings in two natural languages, and are equivalent to the Syntax-Directed Translation Schemata of Aho and Ullman (1969). SCFGs are a subclass of LCFRS where each nonterminal has fan-out two: one span in one language and one span in the other. Binary SCFGs, also known as Inversion Transduction Grammars (ITGs), have no more than two nonterminals on the r.h.s. of a rule, and are the most widely used model in statistical machine translation.

Synchronous parsing with traditional tabular methods for ITG is $O(n^6)$, as each of the three nonterminals in a rule has fan-out of two. ITGs, unfortunately, do not satisfy Condition 4.1, and therefore we have to use the algorithm in Figure 5. Still, just like with TAG, each rule combines two nonterminals of fan-out two using two combination points. Thus, $d = 2$, and we achieve a bound of $O(n^{2\omega+1})$ for ITG, which is $O(n^{5.76})$ using current state of the art for matrix multiplication.

We achieve even greater gains for the case of multilanguage synchronous parsing. Generalizing ITG to allow two nonterminals on the righthand side of a rule in each of k languages, we have an LCFRS with fan-out k. Traditional tabular parsing has an asymptotic complexity of $O(n^{3k})$, while our algorithm has the complexity of $O(n^{\omega k+1})$.

Another interesting case of a synchronous formalism that our algorithm improves the best-well known result for is that of binary synchronous TAGs (Shieber and Schabes 1990) – i.e. a TAG in which all auxiliary trees are binary. This formalism can be reduced to a binary LCFRS. A tabular algorithm for such grammar has the asymptotic complexity of $O(n^{12})$. With our algorithm, $d = 4$ for this formalism, and as such its asymptotic complexity in that case is $O(n^{9.52})$.

7. Discussion and Open Problems

In this section, we discuss some extensions to our algorithm and open problems.

7.1 Turning Recognition into Parsing

The algorithm we presented focuses on recognition: given a string and a grammar, it can decide whether the string is in the language of the grammar or not. From an application perspective, perhaps a more interesting algorithm is one that returns an actual derivation tree, if it identifies that the string is in the language.

It is not difficult to adapt our algorithm to return such a parse, without changing the asymptotic complexity of $O(n^{\omega d+1})$. Once the transitive closure of T is computed, we can backtrack to find such parse, starting with the start symbol in a cell spanning the whole string. When we are in a specific cell, we check all possible combination points (there are d of those) and nonterminals, and if we find such pairs of combination points and nonterminals that are valid in the chart, then we backtrack to the corresponding cells. The asymptotic complexity of this post-processing step is $O(n^{d+1})$, which is less than $O(n^{\omega d})$ ($\omega > 2$, $d > 1$).

This post-processing step corresponds to an algorithm that finds a parse tree, *given* a pre-calculated chart. If the chart was not already available when our algorithm finishes, the asymptotic complexity of this step would correspond to the asymptotic complexity of a naïve tabular parsing algorithm. It remains an open problem to adapt our algorithm to *probabilistic parsing*, for example – finding the highest scoring parse given a probabilistic or a weighted LCFRS (Kallmeyer and Maier 2010). See more details in §7.3.

7.2 General Recognition for Synchronous Parsing

Similarly to LCFRS, the rank of an SCFG is the maximal number of nonterminals that appear in the right-hand side of a rule. Any SCFG can be binarized into an LCFRS grammar. However, when the SCFG rank is arbitrary, this means that the fan-out of the LCFRS grammar can be larger than 2. This happens because binarization creates intermediate nonterminals that span several substrings, denoting binarization steps of the rule. These substrings are eventually combined into two spans, to yield the language of the SCFG grammar (Huang et al. 2009).

Our algorithm does not always improve the asymptotic complexity of SCFG parsing over tabular methods. For example, Figure 6 shows the combination of spans for the rule $S \rightarrow A\,B\,C\,D,\ B\,D\,A\,C$, along with a binarization into three simpler LCFRS rules. A naïve tabular algorithm for this rule would have the asymptotic complexity of $O(n^{10})$, but the binarization shown in Figure 6 reduces this to $O(n^8)$. Our algorithm gives a complexity of $O(n^{9.52})$, as the second step in the binarization shown consists of a rule with $d = 4$.

7.3 Generalization to Weighted Logic Programs

Weighted logic programs (WLPs) are declarative programs, in the form of Horn clauses similar to those that Prolog uses, that can be used to formulate parsing algorithms such as CKY and other types of dynamic programming algorithms or NLP inference algorithms (Eisner, Goldlust, and Smith 2005; Cohen, Simmons, and Smith 2011).

For a given Horn clause, WLPs also require a "join" operation that sums (in some semiring) over a set of possible values in the free variables in the Horn clauses. With CKY, for example, this sum will be performed on the mid-point concatenating two spans. This join operation is also the type of operation we address in this paper (for LCFRS) in order to improve their asymptotic complexity.

It remains an open question to see whether we can generalize our algorithm to arbitrary weighted logic programs. In order to create an algorithm that takes as input a weighted logic program (and a set of axioms) and "recognizes" whether the goal is achievable, we would need to have a generic way of specifying the set N, which was specialized to LCFRS in this case. Not only that, we would have to specify N in such a way that the asymptotic complexity of the WLP would improve over a simple dynamic programming algorithm (or a memoization technique).

In addition, in this paper we focus on the problem of recognition and parsing for unweighted grammars. Benedí and Sánchez (2007) showed how to generalize Valiant's algorithm in order to compute inside probabilities for a PCFG and a string. Even if we were able to generalize our addressing scheme to WLPs, it remains an open question to see whether we can go beyond recognition (or unweighted parsing).

7.4 Relation to Multiple Context-Free Grammars

Nakanishi et al. (1998) developed a matrix multiplication parsing algorithm for multiple context-free grammars (MCFGs). When these grammars are given in a binary form, they can be reduced to binary LCFRS. Similarly, binary LCFRS can be reduced to binary MCFGs. The algorithm that Nakanishi et al. develop is simpler than ours, and does not directly tackle the problem of transitive closure for LCFRS. More specifically, Nakanishi et al. multiply a seed matrix such as our T by itself in several steps, and then follow up with a copying operation between equivalent cells. They repeat this n times, where n is the sentence length. As such, the asymptotic complexity of their algorithm is identical for both balanced and unbalanced grammars, a distinction they do not make.

The complexity analysis of Nakanishi et al. is different than ours, but in certain cases, yields identical results. For example, if $\phi(a) = f$ for all $a \in \mathcal{N}$, and the grammar is balanced, then both our algorithm and their algorithm give a complexity of $O(n^{2\omega d+1})$. If the grammar is unbalanced, then our algorithm gives a complexity of $O(n^{2\omega d})$, while the asymptotic complexity of their algorithm remains $O(n^{2\omega d+1})$. As such, Nakanishi et al.'s algorithm does not generalize Valiant's algorithm – its asymptotic complexity for context-free grammars is $O(n^{\omega+1})$ and not $O(n^{\omega})$.

Nakanishi et al. pose in their paper an open problem, which loosely can be reworded as the problem of finding an algorithm that computes the transitive closure of T without the extra $O(n)$ factor that their algorithm incurs. In our paper, we provide a solution to this open problem for the case of unbalanced grammars. The core of the solution lies in the matrix multiplication copying mechanism described in §4.2.

7.5 Optimal Binarization Strategies

The two main grammar parameters that affect the asymptotic complexity of parsing with LCFRS (in their general form) are the fan-out of the nonterminals and the rank of the rules. With tabular parsing, we can actually refer to the parsing complexity of a *specific rule in the grammar*. Its complexity is $O(n^p)$, where the parsing complexity p is the total fan-out of all nonterminals in the rule. For binary rules of the form $a \rightarrow b\,c$, $p = \varphi(a) + \varphi(b) + \varphi(c)$.

To optimize the tabular algorithm time complexity of parsing with a binary LCFRS, equivalent to another non-binary LCFRS, we would want to minimize the time complexity it takes to parse each rule. As such, our goal is to minimize $\varphi(a) + \varphi(b) + \varphi(c)$ in the resulting binary grammar. Gildea (2011) has shown that this metric corresponds to the tree width of a dependency graph which is constructed from the grammar. It is not known whether finding the optimal binarization of an LCFRS is an NP-complete problem, but Gildea (2011) shows that a polynomial time algorithm would imply improved approximation algorithms for the treewidth of general graphs.

In general, the optimal binarization for tabular parsing may not by the same as the optimal binarization for parsing with our algorithm based on matrix multiplication. In order to optimize the complexity of our algorithm, we want to minimize d, which is the maximum over all rules $a \rightarrow b\,c$ of $d(a \rightarrow b\,c) = \max\{\varphi(a) + \varphi(b) - \varphi(c), \varphi(a) - \varphi(b) + \varphi(c), -\varphi(a) + \varphi(b) + \varphi(c)\}$. For a fixed binarized grammar, d is always less than p, the tabular parsing complexity, and, hence, the optimal d^* over binarizations of an LCFRS is always less than the optimal p^* for tabular parsing. However, whether any savings can be achieved with our algorithm depends on whether $\omega d^* < p^*$, or $\omega d^* + 1 < p^*$ in the case of balanced grammars. Our criterion does not seem to correspond closely to a well-studied graph-theoretic concept such a treewidth, and it remains an open problem to find an efficient algorithm that minimizes this definition of parsing complexity.

It is worth noting that $d(a \rightarrow b\,c) \geq \frac{1}{3}\left(\varphi(a) + \varphi(b) + \varphi(c)\right)$. As such, this gives a lower bound on the time complexity of our algorithm relative to tabular parsing using the same binarized grammar. If $O(n^{t_1})$ is the asymptotic complexity of our algorithm, and $O(n^{t_2})$ is the asymptotic complexity of a tabular algorithm, then $\dfrac{t_1}{t_2} \geq \dfrac{\omega}{3} > 0.79$.

8. Conclusion

We described a parsing algorithm for binary linear context-free rewriting systems that has the asymptotic complexity of $O(n^{\omega d+1}$ where $\omega < 2.38$, d is the "contact rank" of the grammar (the maximal number of combination points in the rules in the grammar) and n is the parsed string length. Our algorithm has the asymptotic complexity of $O(n^{\omega d})$ for a subset of binary

Symbol	Description	1st mention
$M(n)$	The complexity of Boolean $n \times n$ matrix multiplication	§1
ω	Best well-known complexity for $M(n)$, $M(n) = O(n^{\omega})$	§1
$[n]$	Set of integers $\{1, \ldots, n\}$	§2
$[n]_0$	$[n] \cup \{0\}$	§2
$\mathcal{N}$	Nonterminals of the LCFRS	§2
$\mathcal{T}$	Terminal symbols of the LCFRS	§2
$\mathcal{V}$	Variables that denote spans in grammar	§2
$\mathcal{R}$	Rules in the LCFRS	§2
a,b,c	Nonterminals	§2
f	Maximal fan-out of the LCFRS	Eq. 2
$\varphi(a)$	Fan-out of nonterminal a	§2
y	Denoting a variable in $\mathcal{V}$ (potentially subscripted)	§2
T	Seed matrix	§3
N, $N(d)$	Set of indices for addresses in the matrix	Eq. 3
i, j	Indices for cells in T. $i, j \in N$	§4.1
d	Grammar contact rank	§4.1
M	T_{ij} is a subset of M	§4.1
$\mathrm{FromRow}$,ToRow	Copying symbols for rows	§4.1
ToCol,$\mathrm{FromCol}$	Copying symbols for columns	§4.1
$\alpha(i)$	The set of spans taken from matrix coordinate i	§4.1
n	Length of sentence to be parsed	§1
$<$	Strict partial order between the set of indices of T	§4.1
$m(i, j)$	Merged sorted sequence of $\alpha(i) \cup \alpha(j)$, divided into pairs	§4.1
$\mathrm{setdiff}(A, B)$	Set difference between two sets A and B	Figure 3
$\mathrm{remove}(v, x)$	Removal of x from a sequence v	Figure 3
$\mathrm{insert}(v, x)$	Insertion of x in a sequence v	§4.5
Π	Copying operator	§4.5

Table 1
Table of notation symbols used in this paper.

LCFRS which are “unbalanced.” Our result generalizes the algorithm of Valiant (1975), and also reinforces existing results about mildly context-sensitive parsing for tree adjoining grammars (Rajasekaran and Yooseph 1998). Our result also implies that inversion transduction grammars can be parsed in time $O(n^{2\omega+1})$ and that synchronous parsing with k languages has the asymptotic complexity of $O(n^{\omega k+1})$ where k is the number of languages.

A. Notation

Table 1 gives a table of notation for symbols used throughout this paper.

References

Abboud, Amir, Arturs Backurs, and Virginia Vassilevska Williams. 2015. If the current clique algorithms are optimal, so is valiant’s parser. *arXiv preprint arXiv:1504.01431*.

Aho, Alfred V. 1968. Indexed grammars – an extension of context-free grammars. *Journal of the ACM (JACM)*, 15(4):647–671.

Aho, Alfred V. and Jeffery D. Ullman. 1969. Syntax directed translations and the pushdown assembler. *Jounral of Computer and System Sciences*, 3:37–56.

Benedí, José-Miguel and Joan-Andreu Sánchez. 2007. Fast stochastic context-free parsing: A stochastic version of the valiant algorithm. In *Pattern Recognition and Image Analysis*. Springer, pages 80–88.

Cocke, John and Jacob T. Schwartz. 1970. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

Cohen, Shay B., Giorgio Satta, and Michael Collins. 2013. Approximate pcfg parsing using tensor decomposition. In *Proceedings of NAACL*.

Cohen, Shay B., Robert J. Simmons, and Noah A. Smith. 2011. Products of weighted logic programs. *Theory and Practice of Logic Programming*, 11(2–3):263–296.

Coppersmith, D. and S. Winograd. 1987. Matrix multiplication via arithmetic progressions. In *Proceedings of the 19-th annual ACM conference on Theory of computing*, pages 1–6.

Dunlop, Aaron, Nathan Bodenstab, and Brian Roark. 2010. Reducing the grammar constant: an analysis of cyk parsing efficiency. Technical report, Technical report CSLU-2010-02, OHSU.

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Eisner, Jason, Eric Goldlust, and Noah A. Smith. 2005. Compiling Comp Ling: Practical weighted dynamic programming and the Dyna language. In *Proceedings of HLT-EMNLP*, pages 281–290.

Eisner, Jason and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 457–464. Association for Computational Linguistics.

Gazdar, Gerald. 1988. *Applicability of indexed grammars to natural languages*. Springer.

Gildea, Daniel. 2011. Grammar factorization by tree decomposition. *Computational Linguistics*, 37(1):231–248.

Huang, Liang, Hao Zhang, Daniel Gildea, and Kevin Knight. 2009. Binarization of synchronous context-free grammars. *Computational Linguistics*, 35(4):559–595.

Joshi, Aravind K and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*. Springer, pages 69–123.

Kallmeyer, Laura. 2010. *Parsing Beyond Context-Free Grammars*. Cognitive Technologies. Springer.

Kallmeyer, Laura and Wolfgang Maier. 2010. Data-driven parsing with probabilistic linear context-free rewriting systems. In *Proceedings of COLING*.

Kasami, Tadao. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab.

Le Gall, François. 2014. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA. ACM.

Nakanishi, Ryuichi, Keita Takada, Hideki Nii, and Hiroyuki Seki. 1998. Efficient recognition algorithms for parallel multiple context-free languages and for multiple context-free languages. *IEICE TRANSACTIONS on Information and Systems*, 81(11):1148–1161.

Pollard, Carl J. 1984. *Generalized Phrase Structure Grammars, Head Grammars and Natural Languages*. Ph.D. thesis.

Rajasekaran, Sanguthevar and Shibu Yooseph. 1998. TAL parsing in $O(M(n^2))$ time. *Journal of Computer and System Sciences*, 56:83–89.

Raz, Ran. 2002. On the complexity of matrix product. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 144–151. ACM.

Satta, Giorgio. 1992. Recognition of linear context-free rewriting systems. In *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pages 89–95. Association for Computational Linguistics.

Satta, Giorgio. 1994. Tree-adjoining grammar parsing and boolean matrix multiplication. *Computational linguistics*, 20(2):173–191.

Shieber, Stuart M and Yves Schabes. 1990. Synchronous tree-adjoining grammars. In *Proceedings of the 13th conference on Computational linguistics-Volume 3*, pages 253–258. Association for Computational Linguistics.

Steedman, Mark. 2000. *The Syntactic Process*. Language, speech, and communication. MIT Press, Cambridge (Mass.), London.

Strassen, V. 1969. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356.

Valiant, Leslie G. 1975. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308–315.

Vijay-Shanker, K. and David Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational linguistics*, 23(3):377–403.

Younger, Daniel H. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2).